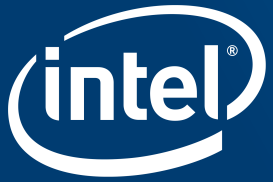


Communicators and Windows and Threads, Oh My!

James Dinan
Extreme Scale Software Pathfinding Team

Celebrating 25 Years of MPI
September 25, 2017

Legal Notices and Disclaimers

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

No computer system can be absolutely secure.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Intel, the Intel logo, Xeon and Xeon Phi and others are trademarks of Intel Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

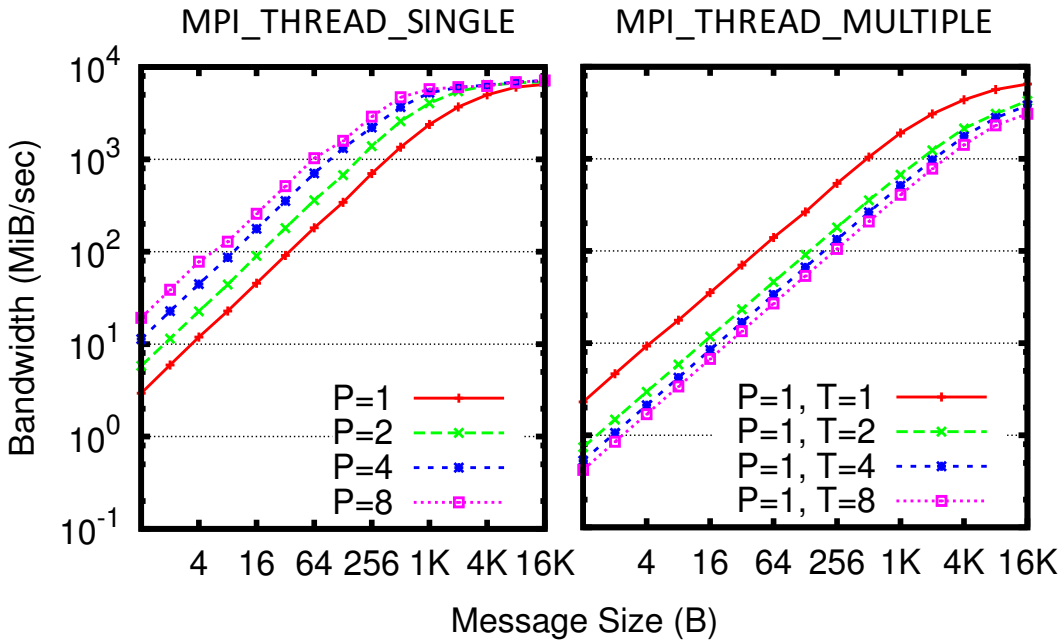
Notice revision #20110804

Let's be honest about MPI_THREAD_MULTIPLE

Historically, performance has looked like a bit like this

Data is old and improvements are being made, but still …

- Threads interfere in MPI semantics (ordering, etc.)
- MPI is unaware of threads
- Tools are unaware of threads



Node: 2x 12-core 2.7 GHz Intel® Xeon® E5-2697
Fabric: Mellanox* InfiniBand* FDR, 2-level fat-tree
Intel® MPI Library v4.1.3, no modifications

[SC14] "Enabling efficient multithreaded MPI communication through a library-based implementation of MPI endpoints," Sridharan et al.

Multithreaded Processes, Coming Soon to A NIC Near You!

NICs provisioned to support multiple cores

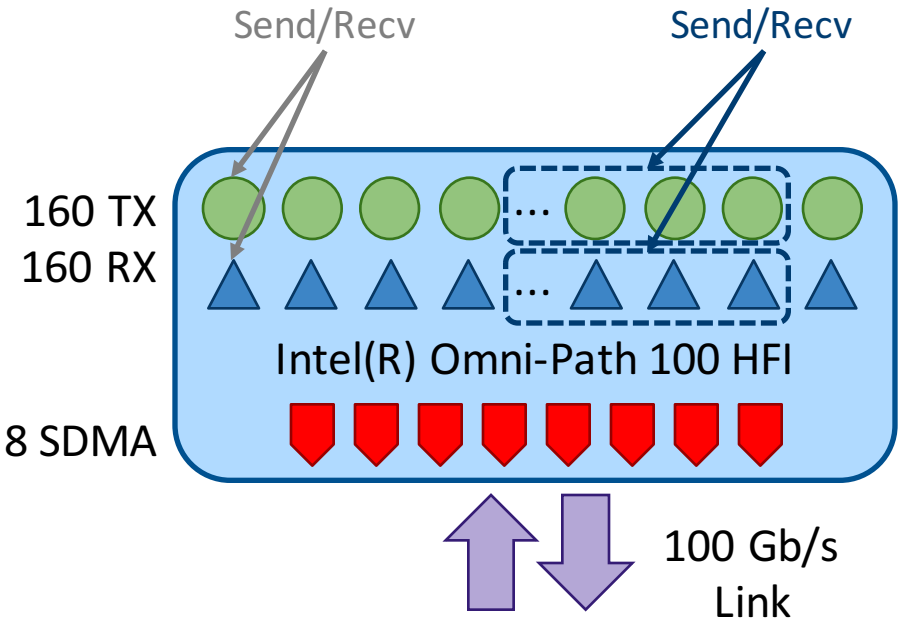
- Must use multiple TX/RX contexts to fully utilize NIC's TX and RX capabilities

Current solution: Run multiple processes

- Restricts application's choice of threads per process, limits thread scaling

Desired: One process drives multiple contexts

- Ideally, map threads to contexts
- Emergent networking stack support
- Challenge: Process-level ordering, sync.
  - Point-to-point, RMA, etc.



Communicator Info to the Rescue?

Ratified for next version of MPI specification:

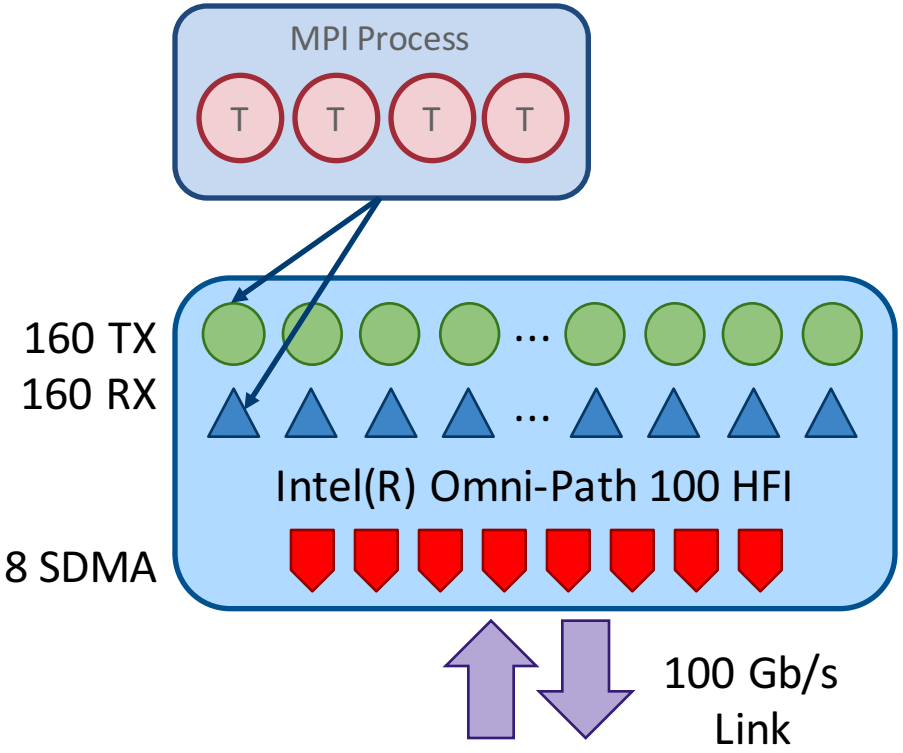
- Allow info hints to convey assertions about behavior of application
- Four new info keys for MPI communicators
  - `mpi_assert_no_any_tag` – TX/RX: Hash on tag
  - `mpi_assert_no_any_source` – RX: Hash on source (TX can always hash on dest.)
  - `mpi_assert_exact_length` – Optimize large message protocols
  - `mpi_assert_allow_overtaking` – TX: Hash on tag/source

Enable single process to distribute (hash) traffic across TX/RX contexts

- But, at the expense of disabling certain MPI semantics
- TX/RX contexts are still shared, not privatized to threads

Typical Mapping of MPI_THREAD_MULTIPLE

Inefficient, because multiple threads drive single TX/RX context pair

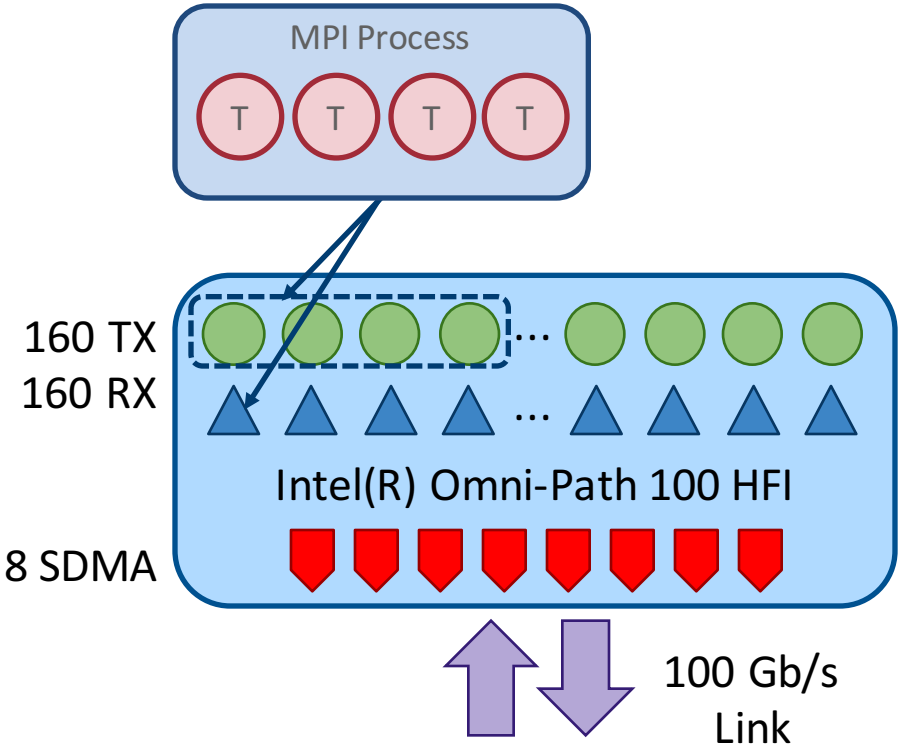


Better …

Processes use multiple TX contexts

MPI 3.1

- Hash on recipient



Even Better …

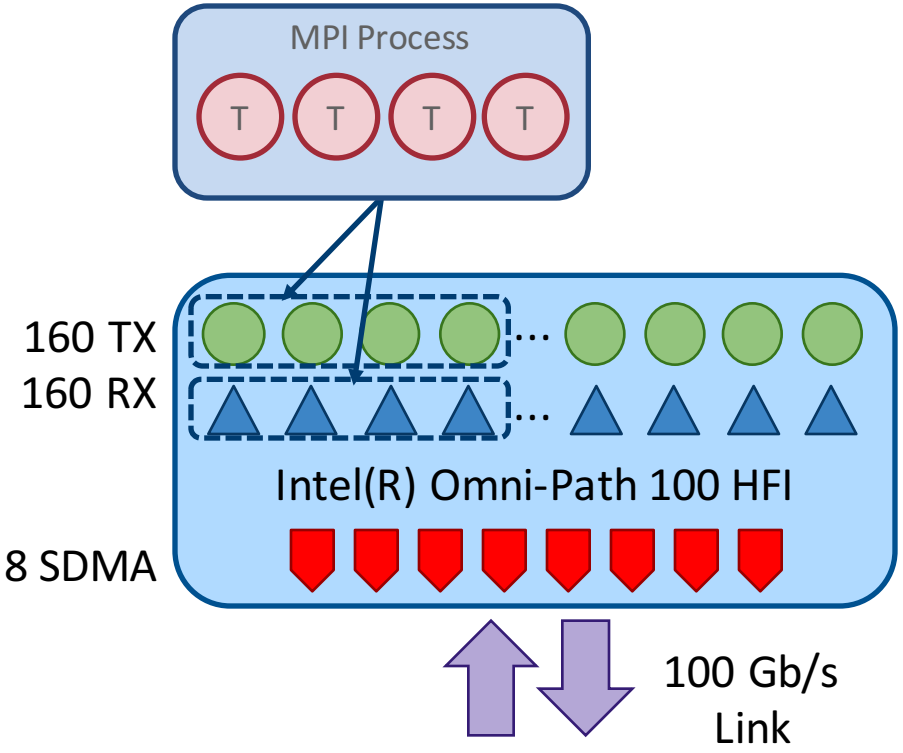
Process uses multiple TX/RX contexts

MPI 3.1

- Communicator per thread

MPI 3.next

- Threads share communicator with info assertions (e.g. no wildcards)



Best …

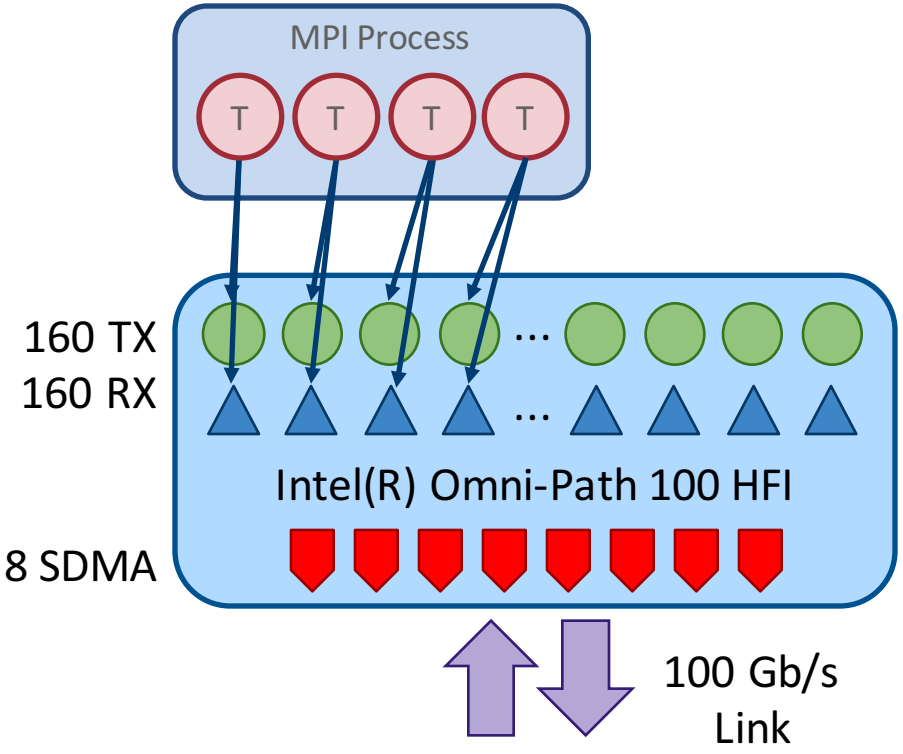
Assignment of threads to TX/RX ctx.

- Eliminate synchronization overheads
- Improve latency, small message throughput

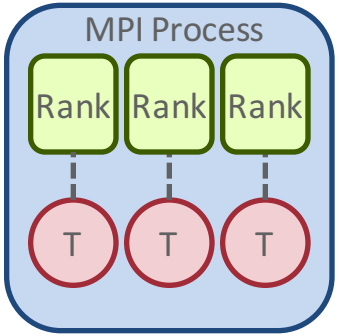
Communicator per thread

- Plus an info hint that the communicator is private to the thread

Endpoints (!!)



History of MPI Endpoints



Static endpoints, proposed 8-28-2011 by Marc Snir

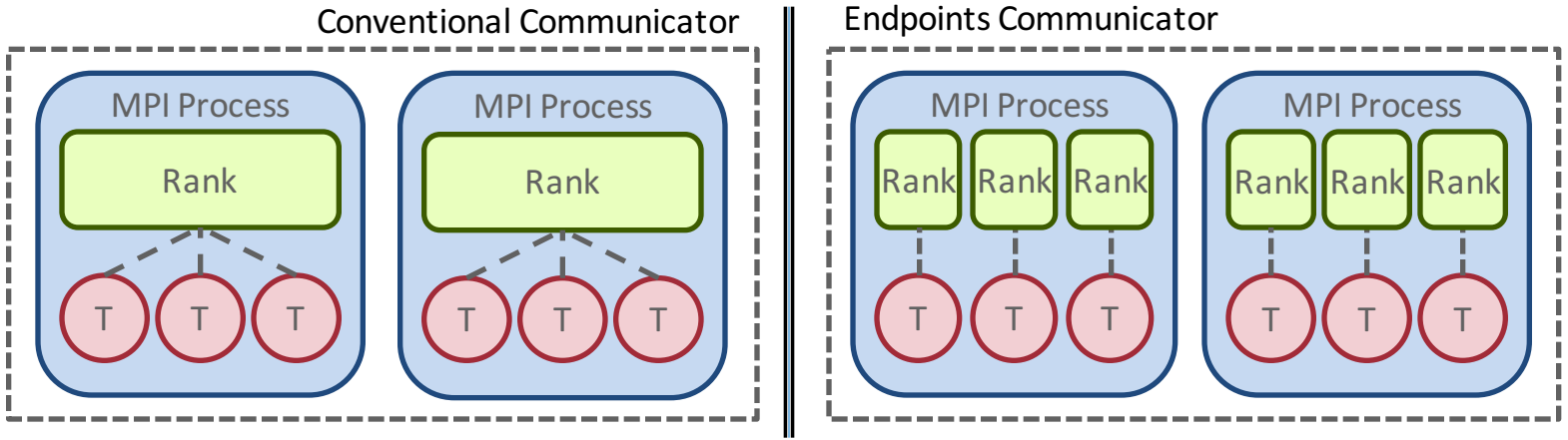
MPI_Comm_create_endpoints, proposed 7-12-2013 by Jim Dinan

- Proposal draft: https://github.com/mpi-forum/mpi-issues/issues/56

Publications:

- [EuroMPI '13] *Enabling MPI Interoperability Through Flexible Communication Endpoints*. James Dinan, Pavan Balaji, David Goodell, Douglas Miller, Marc Snir, and Rajeev Thakur.
- [IJHPCA '14] *Enabling Communication Concurrency Through Flexible MPI Endpoints.* James Dinan, Ryan E. Grant, Pavan Balaji, David Goodell, Douglas Miller, Marc Snir, and Rajeev Thakur.
- [SC '14] *Enabling Efficient Multithreaded MPI Communication Through a Library-Based Implementation of MPI Endpoints*. Srinivas Sridharan, James Dinan, and Dhiraj Kalamkar.

MPI Endpoints Proposal



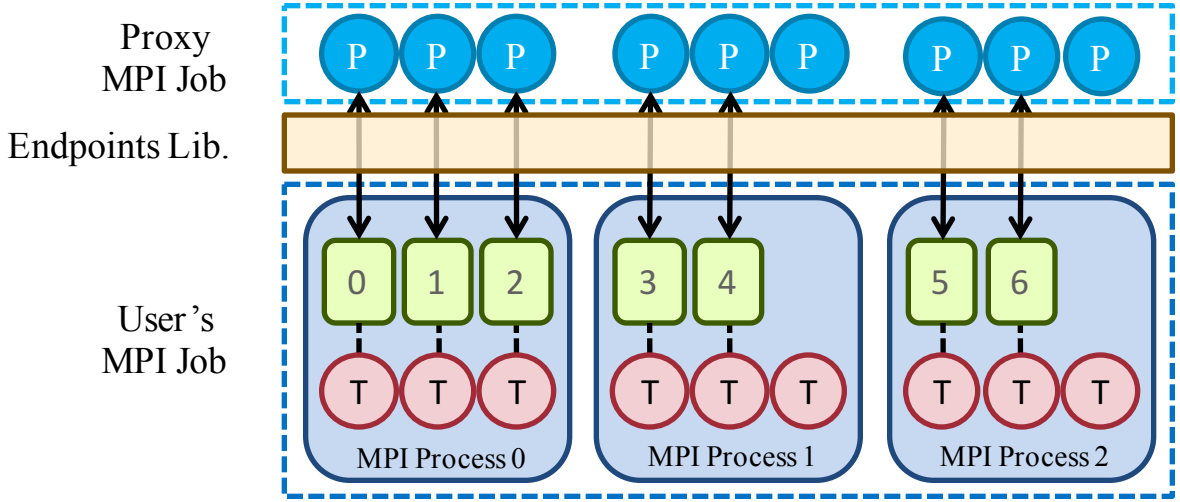
A rank is an abstract entity representing an MPI communication "endpoint"

- Set of resources that supports the execution of MPI operations

Proposal: Fork new ranks from existing ranks in parent communicator to enable many-to-one mapping

- MPI_Comm_create_endpoints(MPI_Comm parent, int num_ep, MPI_Info info, MPI_Comm ep_comm[])
- Endpoint ranks behave like MPI processes (progress, matching, ordering rules)

EP-Lib: MPI Endpoints Library

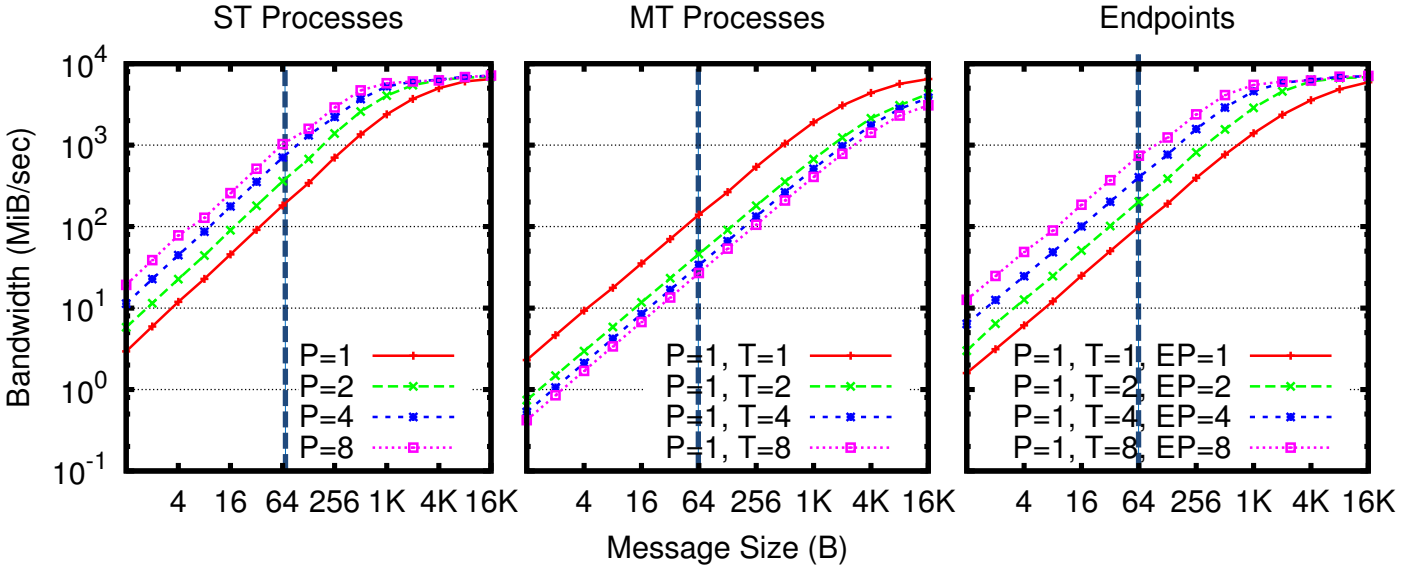


Spawn background MPI job (may oversubscribe), implement endpoint ranks using MPI processes

Endpoints library forwards commands from user job to proxy job

- Proxy process performs MPI operation on behalf of user endpoint rank
- POSIX Shared memory coordination between user and proxy job

Impact on Throughput



Single threaded (ST), multithreaded (MT), and endpoints cases

- Two nodes, increase number of process or threads

Node: 2x 12-core 2.7 GHz Intel® Xeon® E5-2697
Fabric: Mellanox* InfiniBand* FDR, 2-level fat-tree
Intel® MPI Library v4.1.3, no modifications

Uni-directional BW at 64B messages, 8 cores (ST = 1029 (100%); MT = 27 (2.6%); EP = 742 (72%) MiB/sec)

Can Communicator Info Help?

MPI_TAG_ORDERING_KEY (integer) = *value*

The low *value* bits of tags used in point-to-point communication on the given communicator represent an ordering key at the receiver (e.g. receiver thread ID). When this info key is provided, tags are of the form *key* + (*tag* << *value*). The full tag must be less than or equal to the value of MPI_TAG_UB. Point-to-point communication operations whose tag contain the same ordering key obey the nonovertaking semantic. Messages with different ordering keys have no relative ordering.

What About Threads and RMA?

Synchronization is at the window level

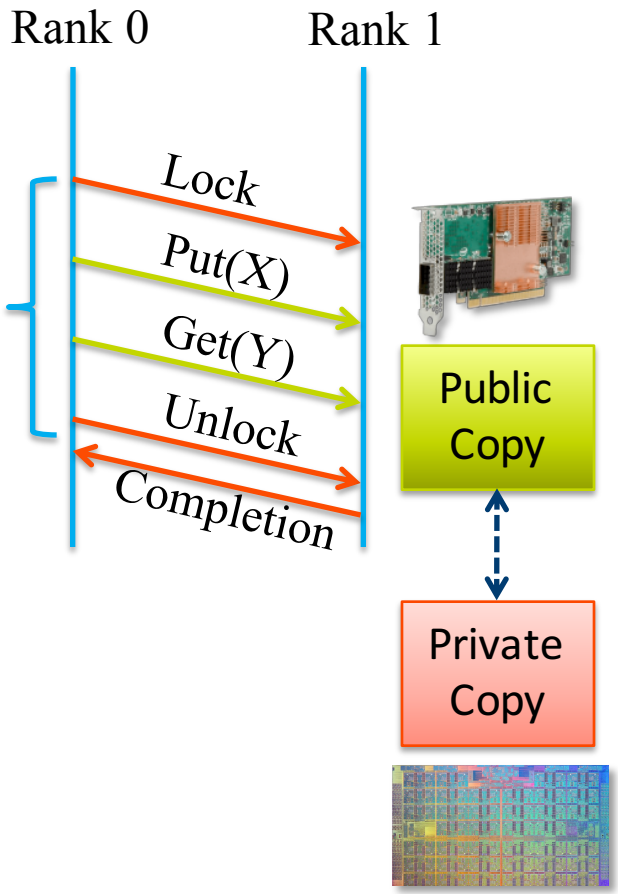
Active target is reasonable, collective on process

Passive target synchronization is hard

- Lock/unlock, flush, sync
- Source of thread interference
- Easy to violate sync. rules

Solution is multiple overlapping windows

- Provides isolation
- Still easy (easier?) to violate sync. rules



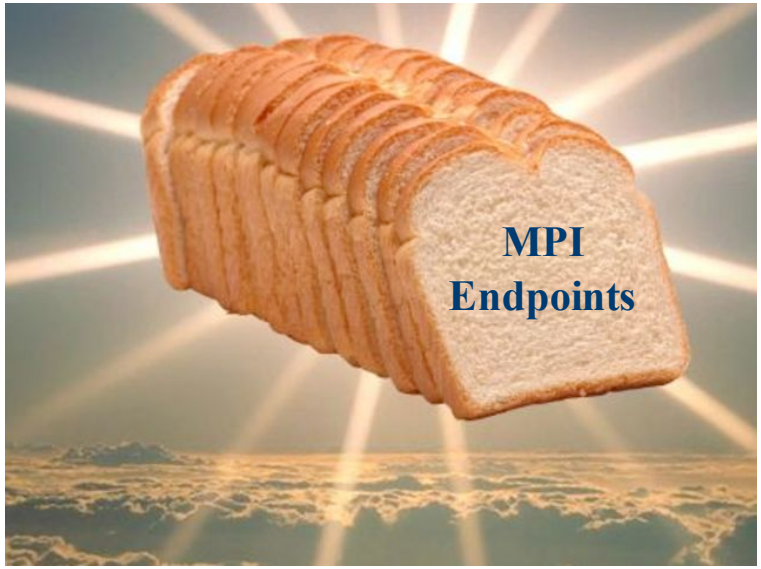
Wrap-Up

There are semantic challenges associated with MPI_THREAD_MULTIPLE that implementation cleverness can't remedy

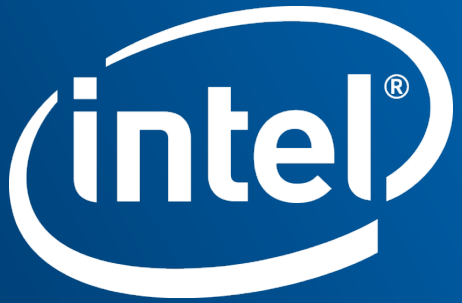
- Info assertions might provide duct tape solution for point-to-point
- There are still challenges to interoperability of RMA and other interfaces with threads

Endpoints are the best thing since sliced bread

- Solve mapping problems without restricting MPI semantics



Credit: https://priceconomics.com/the-invention-of-sliced-bread/


intel®